Hyperparameter Learning for Deep Learning-Based Recommender Systems

Di Wu, *Member, IEEE*, Bo Sun, and Mingsheng Shang

Abstract—Deep learning (DL)-based recommender system (RS), particularly for its advances in the recent five years, has been startling. It reshapes the architectures of traditional RSs by lifting their limitations in dealing with data sparsity and cold-start issues. Yet, the performance of DL-based RS, like many other DL-based intelligent systems, heavily relies on selecting hyperparameters. Unfortunately, the most common selection approach is still Grid Search that requires numerous computational resources and human efforts. Motivated by this, this paper proposes a general hyperparameter optimization framework, named DE-Opt, which can be applied to most existing DL-based RSs seamlessly. The main idea of DE-Opt is to incorporate differential evolution (DE) into the model training process of a DL-based RS at a layer-wise granularity to simultaneously auto-learn its two key hyperparameters, namely, the learning rate η and the regularization coefficient λ. As a result, the system performance in terms of recommendation accuracy and computational efficiency is uplifted. Experimental results on three benchmark datasets substantiate that: 1) DE-Opt is compatible with the most recent advancements of DL-based RS to automate their hyperparameter-tuning processes, and 2) DE-Opt excels among its state-of-the-art hyperparameter-optimization competitors in terms of both higher learning performance and lower runtime.

Index Terms—Deep learning, differential evolution, grid search, hyperparameter tuning, online services, recommender systems

1 Introduction

Recommender System (RS) has become an indispensable building block in online services for business-boosting and user-experience elevation [1], [2]. Traditional RS tends to yield inferior performance in resolving the data sparsity and cold-start problems without sacrificing recommendation quality [1], [2], [4]. Deep learning (DL) has been advocated as a plausible solution to the issue, thanks to its strong capability of modeling complex and non-linear user-item interactions [5], [6]. In recent DL-based RSs, superior recommendation performance is observed [3], [5], [33], [34].

Despite the superiority, existing DL-based RSs mainly suffer from a common drawback—the tedious process of hyperparameter tuning [7], [34]. On the one hand, due to the black-box nature of DL model architectures, the hyperparameters in a DL-based RS directly decide the system properties in both training and trained states, i.e., how fast the RS can converge and how well the converged equilibrium optimizes the designed learning objective [38], [39]. On the other hand, the ubiquity of DL-based RSs exacerbates this nature. Various regularization terms have been engineered to incorporate a diversity of side information from the items and users (e.g., textual description of items and geographical relations among users) in a wide range of recommendation applications. Such side information is the key to making the DL-based RS great [7], [8], [9], [34], [38], [39]. While, each regularization term correspondingly added to the learning objective introduces a new hyperparameter, making the tuning effort even more cumbersome. A concrete example that substantiates the impact of hyperparameter tuning in a DL-based RS is presented as follows.

Example 1. A typical DL-based recommendation model is trained by optimizing an objective taking the form of $L(\Theta) = J(\Theta) + \lambda\Omega(\Theta)$ [5], where Θ parameterizes the RS and $J(\Theta)$ and $\Omega(\Theta)$ denote the loss and regularization terms, respectively. Two hyperparameters exist in this example, as shown in the top panel of Fig. 1. The first is λ in the learning objective that balances the scale between the loss and regularization terms. The second is the learning rate, denoted by η, which decides how fast the RS is optimized based on stochastic oracles (e.g., gradients or Hessians) in an iterative fashion [15], [16]. To test the significance of λ and η, we hereby evaluate two representative DL-based RS models, i.e., AutoRec [22] (rating prediction) and NeuMF [14] (item ranking), on the MovieLens 1M dataset downloaded from MovieLens[1]. Their performance is presented in the bottom panel of Fig. 1. We observe that the variation of λ and η leads to the extensive changes of model performance, where

- *Di Wu is with the College of Computer and Information Science, Southwest University, Chongqing 400715, China, and also with the Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing 400714, China. E-mail: wudi.cigit@gmail.com.*
- *Bo Sun and Mingsheng Shang are with the Chongqing Key Laboratory of Big Data and Intelligent Computing, Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing 400714, China, and also with the Chongqing School, University of Chinese Academy of Sciences, Chongqing 400714, China.. E-mail: {sunbo, msshang}@cigit.ac.cn.*

Manuscript received 7 August 2022; revised 25 December 2022; accepted 3 January 2023. Date of publication 16 January 2023; date of current version 8 August 2023.
This work was supported in part by the National Natural Science Foundation of China under Grants 62072429, 62176070, 62293500, and 62293501, and in part by the Key Cooperation Project of Chongqing Municipal Education Commission under Grants HZ2021008 and HZ2021017.
(Corresponding Author: Mingsheng Shang.)
Recommended for acceptance by M. Hauswirth.
Digital Object Identifier no. 10.1109/TSC.2023.3234623

1. https://grouplens.org/datasets/movielens/

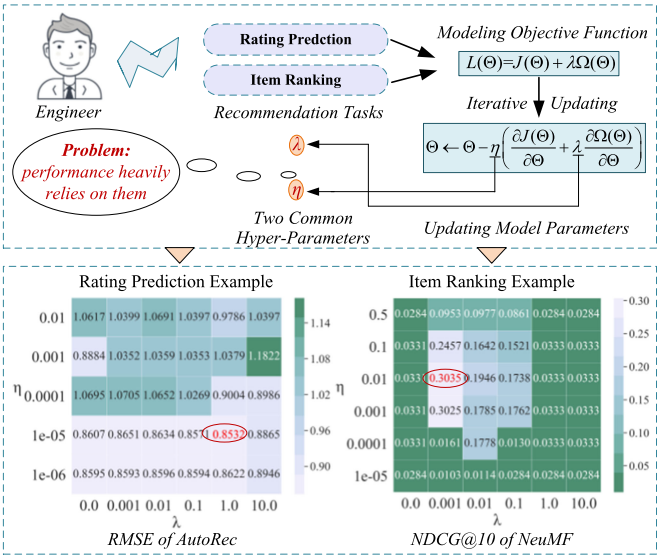


Fig. 1. An example of hyperparameters tuning: AutoRec and NeuMF are highly sensitive to η and λ (lower RMSE and higher NDCG@10 are better).

AutoRec and NeuMF respectively achieve different root mean squared error (RMSE) [22] and normalized discounted cumulative gain (NDCG, NDCG@10 denotes top 10) [14]. The lighter the color, the better the performance. We note that the optimal models are resulted from different values of $\boldsymbol{\lambda}$ and $\boldsymbol{\eta}$ (colored in red).

Example 1 reveals how the performance of a DL-based RS is heavily dependent on an ad-hoc setting of hyperparameters, which has been manifested in a variety of DL-based RS designs [7], [8], [9], [35]. Thus far, to tune the RS hyperparameters, the most common approach is still Grid Search [10], [15], where the optimal hyperparameters are searched in predefined intervals. This search inevitably incurs a combinatorial complexity, which can consume numerous computational resources and human efforts. Therefore, alternatives to Grid Search are desired for efficient and effective hyperparameter searching.

Motivated by this, this paper proposes a general hyperparameter optimization framework for DL-based RSs based on differential evolution (DE) [18], [19], [20], named DE-Opt. Its key ideas are two-fold as follows. 1) DE-Opt decomposes and controls the RS training process at a finer level of granularity by allowing the hyperparameters to differ across neural layers. This enlarges the search space for optimal hyperparameters yet alleviates the impact of each hyperparameter so as to better the resultant models. 2) DE-Opt frames the hyperparameter tuning task as an optimization problem and embeds it into the RS training framework, so that the model parameters and the hyperparameters of an RS are jointly trained. As such, DE-Opt allows to train the RSs only once (in contrast, Grid Search requires to train RSs repeatedly by trying a different hyperparameter combination per each repeat), thereby substantially improving the efficiency and effectiveness of DL-based RSs by attaining their optimal RS models and hyperparameters all at once.

The specific contributions of this paper include:

- This is the first work to propose the auto-learning of the hyperparameters of DL-based RSs at a layer-wise granularity, where DE is employed to solve the non-convex and combinatorial optimization problems.
- The proposed DE-Opt framework is agnostic to the learning objectives and thus is compatible with most existing DL-based RSs, boosting their training effectiveness and efficiency by making their hyperparameters auto-learned along with the model parameters jointly.
- Theoretical analyses and algorithm designs are provided for the proposed DE-Opt framework.

Extensive experiments on three real-world datasets are conducted. The results verify that our DE-Opt: 1) can significantly improve the state-of-the-art DL-based RSs by making their hyperparameters be auto-learned, and 2) significantly outperforms the state-of-the-art adaptive hyperparameter tuning competitors. Besides, to promote reproducible research, we open-access our code at the following link: https://github.com/Wuziqiao/DE-Opt.

2 Related Work

In a nutshell, the mainstream RS designs can be categorized into two settings [5], [15], [16], i.e., **rating prediction** [36], [37] and **item ranking** (i.e., top-K recommendation) [14]. The pioneering study that implements deep neural networks into rating prediction task is AutoRec [22], wherein an autoencoder architecture realizes collaborative filtering [46]. Since then, a flurry of deep learning (DL)-based rating prediction RS has been proposed. In [23], a rating prediction model with abstractive tips generation (NRT) is devised by combining multilayer perceptron and recurrent neural networks. More recently, [24] proposes a federated rating prediction model (MetaMF) based on meta-learning [38], and [28] proposes a multi-component graph convolutional collaborative filtering method for rating prediction. For item ranking, a classic model is NeuMF [14] which firstly combines matrix factorization and multilayer perceptron. Later, [25] proposes a latent relevant metric learning model (LRML) to learn the correlation between users and items. Graph convolution network (GCN) is also employed to implement item ranking RSs. Proposed by He et al., two state-of-the-art GCN-based RSs, namely, NGCF [21] and LightGCN [27], have been devised to achieve remarkable item ranking recommendation performance.

Nevertheless, these methods all suffer a limitation that their hyperparameters λ and η must be chosen in prior. To be applied in practice, a process of hyperparameter tuning, such as Grid Search [10], [15], has to be tortured. Typically, a combination of multiple candidate hyperparameter sets needs to be tried repeatedly for the RS model. The best set of hyperparameters yielding from a held-out validation dataset is selected. This is a tedious effort that consumes considerable time, manpower, and computational resources.

To alleviate this limitation, several efforts are made to automate the hyperparameter tuning of RSs. A classic study is SGDA [11] which updates the regularization coefficient λ and the RS model parameters via alternating optimization. Inspired by SGDA, λOpt is proposed [8] to tune λ adaptively by enforcing regularization during training, which also reveals that an adaptive regularization at a layer-wise finer level can bring performance benefits. As a parallel

effort, several studies focused on tuning the other significant hyperparameter – the learning rate $\boldsymbol{\eta}$. A representative η-adaptive optimizer is Adam [17], which is a first-order gradient-based stochastic optimizer based on adaptive estimation of lower-order momentum. Later, AMSGrad [12] and AdaMod [13] were proposed to aid its sub-optimal convergence rate and the ill-conditioned initialization of Adam by bounding the regularities of momenta.

Unfortunately, these approaches can only tune a single hyperparameter λ or η at each time, while no one works for both. As such, if one covets to simultaneously tune both λ and η adaptively at once, a compromise method is yielded. For example, a straightforward adaptation is to employ an η-adaptive optimizer (e.g., Adam) to optimize a λ-adaptive model (e.g., λOpt). Such a method, alas, boils down to be a slightly improved Grid Search, leading to the suboptimal pairing of hyperparameter set and the model parameters and ending with inferior learning efficiency and performance.

To counter the issue, this paper presents DE-Opt, which significantly differs from the prior studies in the sense that it can make existing DL-based RSs' λ and η be simultaneously auto-learned at layer-granularity (i.e., each layer is allowed to be equipped with different λ and η pairs) during the training on-the-fly. Moreover, the proposed DE-Opt is a generic framework that is compatible with most existing DL-based RSs to boost their learning performance and training efficiency.

3 Preliminaries

3.1 Symbols and Notations

For clarification, we summarize the main notations employed in Table 1. The complete symbols and notations are deferred to Table S.1 in the Supplementary File, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSC.2023.3234623.

TABLE 1
Symbols and Notations

Symbol	Explanation
$L(\cdot)$, Θ	The objective function and its model parameters, respectively.
$J(\cdot)$, $\Omega(\cdot)$	The loss and regularization terms of $L(\cdot)$, respectively.
$L_{u,j}(\cdot)$, $J_{u,j}(\cdot)$	The state of $L(\cdot)$ and $J(\cdot)$ on $y_{u,j}$, respectively.
W, b	The weight matrix and bias vector sets of N layers, respectively.
l^n	The output vector of n-th layer after activation, $n \in \{12..., N\}$.
W^n, b^n	The weight matrix and bias vector of n-th layer, respectively.
x	Initializing embedding vector.
$f(\cdot)$	The activation function.
Z, D	The observed explicit ratings and training triplets sets of input sparse matrix for rating prediction and item ranking, respectively.
$y_{u,j}$, $\hat{y}_{u,j}$	The observed ground truth of interaction between user u and item j and its corresponding prediction.
$\boldsymbol{\lambda}^n$, $\boldsymbol{\eta}^n$	The regularization coefficient and learning rate of n-th layer, respectively.
Δ^n	The term *w.r.t.* partial derivative at n-th layer.
H, Λ	The fine-grained vectorized η and λ of N layers, respectively.
NP	The total number of individuals of DE.
X_i	The i-th target vector (individual), $i \in \{1, 2, \ldots, NP\}$.
V_i	The mutant vector generated for each target vector X_i.
r_1, r_2, r_3	The randomly chosen number from $\{1, 2, \ldots, NP\}$.
F_i	The scaling factor controlling the scaling of various vectors.
U_i	The new trial vector generated from X_i and V_i.
$perf(\cdot)$	The fitness/performance function.
Γ, Ψ	The testing set and its contained users set, respectively.
K	Cutoff numbers of items in the given ranked list to each user.
$\Gamma_K(u)$	The intersection set between the top-K items generated by the tested model and the testing items of Γ by user u.

3.2 General Learning Paradigm of DL-based RS

Despite its multiple variants, the learning procedure of the DL-based RS can be uniformly framed as follows [24]:

$$\begin{cases} l^1 = f\left(W^1 x + b^1\right), \\ l^n = f\left(W^n l^{n-1} + b^n\right), \\ l^N = f\left(W^N l^{N-1} + b^N\right), \end{cases} \tag{1}$$

where x indicates the initialized (user or item) embedding vector, l^n indicates the output vector of n-th layer after activation, $f(\cdot)$ indicates the activation function, W^n indicates the n-th weight matrix, b^n denotes the n-th bias vector, $n \in \{2, \ldots, N\}$, and l^N indicates the final output of the last layer after activation.

To model a DL-based RS, the design of the objective function $L(\Theta) = J(\Theta) + \boldsymbol{\lambda}\Omega(\Theta)$ is crucial [5], [40], [41]. Specifically, point-wise loss and pair-wise loss are commonly used to formulate the loss term $J(\Theta)$ of rating prediction and item ranking, respectively [5], [15]. For **rating prediction**, a general objective function can be usually formulated with square distance as follows:

$$L_{Rating}(\Theta) = \sum_{(u,j)\in Z} \left(\left(\hat{y}_{u,j} - y_{u,j}\right)^2 | \Theta \right) + \lambda\Omega(\Theta), \tag{2}$$

where $y_{u,j}$ denotes the observed interaction between a user u and an item j. Denoted by $\hat{y}_{u,j}$ is the prediction of the ground-truth $y_{u,j}$, and by Z is the observed explicit rating set of the input matrix which is sparse and high-dimensional in its argument.

For **item ranking**, (2) can not be used for binarized implicit feedback well [26]. Alternatively, the pair-wise with Bayesian Personalized Ranking (BPR) loss [26] is commonly adopted. Its general objective function is formulated as follows:

$$L_{Ranking}(\Theta) = -\sum_{(u,j,j')\in D} \ln\left(f\left(\left(\hat{y}_{u,j} - \hat{y}_{u,j'}\right)|\Theta\right) + \lambda\Omega(\Theta)\right), \tag{3}$$

where $D = \{(u, j, j') \mid y_{u,j} = 1, y_{u,j'} = 0,)\}$ denotes the training triplets set, $\hat{y}_{u,j}$ and $\hat{y}_{u,j'}$ denote the predictions of interactions (u, j) and (u, j'), respectively.

Finally, the training of DL-based recommendation models is to minimize their objective function [27]. To this end,

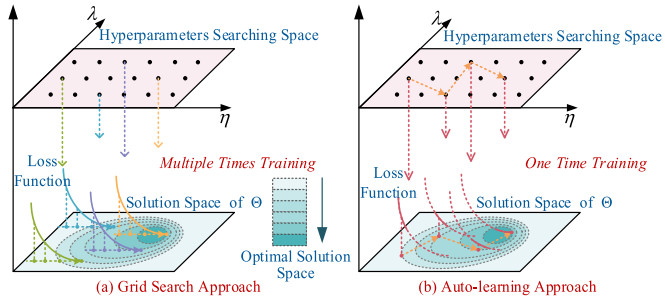


Fig. 2. The differences of hyperparameters tuning between Grid Search and auto-learning approach.

stochastic gradient descent (SGD) [28], [29], [30] is most commonly adopted.

4 The Proposed DE-Opt Optimization Framework

4.1 Motivation of DE-Opt

The performance of a DL-based recommendation model heavily relies on hyperparameters tuning $\boldsymbol{\eta}$ and $\boldsymbol{\lambda}$. Fig. 2 visualizes the differences between the two tuning approaches of Grid Search and auto-learning, where each one has a hyperparameter searching space (candidate values of $\boldsymbol{\eta}$ and $\boldsymbol{\lambda}$) and solution space of Θ (desired model parameters Θ), and the arrows represent the searching paths.

Grid Search. First, $\boldsymbol{\eta}$ and $\boldsymbol{\lambda}$ are initialized with two values from the searching space. After that, they become a pair of fixed values in all the steps of minimizing the loss function. Each pair of $\boldsymbol{\eta}$ and $\boldsymbol{\lambda}$ in the searching space corresponds to a solution in the solution space. Fig. 2a shows that if we want to find the optimal solution of Θ, we need to traverse the complete searching space where each search requires full training steps. As a result, Grid Search either consumes computational resources or requires prior knowledge about selecting appropriate $\boldsymbol{\eta}$ and $\boldsymbol{\lambda}$ candidates.

Auto-Learning. Different from Grid Search, auto-learning allows employing different $\boldsymbol{\eta}$ and $\boldsymbol{\lambda}$ at each training step. Fig. 2b shows that auto-learning only needs one-time training to search for the optimal Θ. Although auto-learned trajectories can greatly reduce computational consumption, there are two key observations regarding the deficiencies of this auto-learning process, which motivate our DE-Opt design as follows.

- At different training stages, the learning rate of gradient descent and strength of regularization should be different. Since the model has not learned much from data at the early training stages, the learning rate should be large and the regularization should be light. In contrast, a small learning rate and strong regularization are required to finely search optimal Θ and avoid overfitting at the end training stages.
- Each layer should have a different learning rate of gradient descent and strength of regularization. During the backpropagation of errors, the gradient may vanish or explode from the last layer to the first layer, preventing to search for the globally optimal Θ. One possible way of alleviating this problem is to set different $\boldsymbol{\eta}$ and $\boldsymbol{\lambda}$ at layer-granularity.

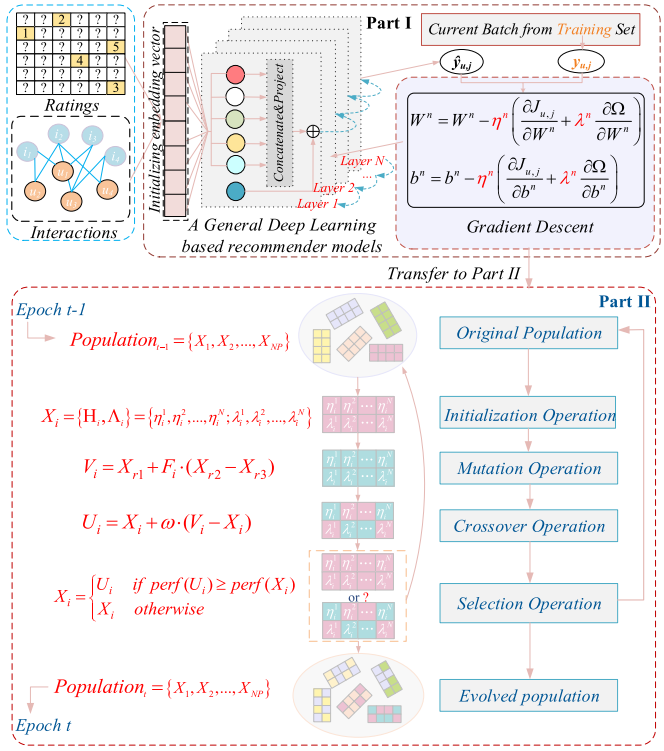


Fig. 3. The Architecture of DE-Opt.

4.2 The Architecture of DE-Opt

Upon the two observations, we aim to build an auto-learning scheme for DL-based recommendation models' $\boldsymbol{\eta}$ and $\boldsymbol{\lambda}$ at layer granularity simultaneously during the training process. Note that this auto-learning problem is discrete. To solve it, we employ differential evolution (DE) [18], [19], [20], which is known for its high efficiency and robustness in obtaining global optima in discrete numerical optimization tasks. In particular, our DE-Opt architectural design, as shown in Fig. 3, consists of two parts. Part I shows how to train a DL-based recommendation model layer by layer, where $\boldsymbol{\lambda}$ and $\boldsymbol{\eta}$ are designed at layer-granularity, i.e., each layer has different $\boldsymbol{\lambda}$ and $\boldsymbol{\eta}$. Part II is to employ DE to simultaneously auto-learn each layer's $\boldsymbol{\lambda}$ and $\boldsymbol{\eta}$ at each epoch, including four operations, i.e., initialization, mutation, crossover, and selection. We present the details of the two building blocks in the following sections.

4.2.1 *Part I: Training a DL-based Recommendation Model With Layer-wise Granular Hyperparameter Control*

To finely control different layers, $\boldsymbol{\lambda}$ and $\boldsymbol{\eta}$ are matched with the number of hidden layers, i.e., model depth, as shown in Table 2. The Λ and H denote the $\boldsymbol{\lambda}$ and $\boldsymbol{\eta}$ vectors, respectively, where $\boldsymbol{\lambda}^n$ and $\boldsymbol{\eta}^n$ denote $\boldsymbol{\lambda}$ and $\boldsymbol{\eta}$ at n-th layer, $n \in \{1, 2, .., N\}$, respectively.

TABLE 2
The Vectorized Hyperparameters of $\boldsymbol{\lambda}$ and $\boldsymbol{\eta}$

Hyperparameters	Layer 1	...	Layer n	...	Layer N
Regularization Coefficient: Λ	$\boldsymbol{\lambda}^1$		$\boldsymbol{\lambda}^n$		$\boldsymbol{\lambda}^N$
Learning Rate: H	$\boldsymbol{\eta}^1$		$\boldsymbol{\eta}^n$		$\boldsymbol{\eta}^N$

Supposing input the observed ground-truth $y_{u,j}$ of interaction (u, j) between user u and item j at the current batch in the training set, we have the current objective function as follows:

$$L_{u,j}(\Theta) = L_{u,j}(W, b) = J_{u,j}(W, b) + \Lambda \odot \Omega(W, b), \tag{4}$$

where $J_{u,j}(\cdot)$ indicates the *loss* function on $y_{u,j}$, W is the weight matrix sets of N layers, b is the bias vector sets of N layers, and $\odot$ indicates the element-wise product between parameters of each layer and Λ. By employing SGD to train (4) [15]–[17], we obtain the training rules for n-th layer's model parameters as follows:

$$W^n \leftarrow W^n - \eta^n \frac{\partial L_{u,i}}{\partial W^n} = W^n - \eta^n \left(\frac{\partial J_{u,j}}{\partial W^n} + \lambda^n \frac{\partial \Omega}{\partial W^n} \right) \tag{5}$$

$$b^n \leftarrow b^n - \eta^n \frac{\partial L_{u,i}}{\partial b^n} = b^n - \eta^n \left(\frac{\partial J_{u,j}}{\partial b^n} + \lambda^n \frac{\partial \Omega}{\partial b^n} \right). \tag{6}$$

Note that in (5) and (6), the terms of $\partial\Omega/\partial W^n$ and $\partial\Omega/\partial b^n$ can be directly computed while that of $\partial J_{u,j}/\partial W^n$, and $\partial J_{u,j}/\partial b^n$ require the chain rule [5], [6], [7] to compute. Then, we can compute $\partial J_{u,j}/\partial W^n$ and $\partial J_{u,j}/\partial b^n$ as follows:

$$\begin{aligned} \frac{\partial J_{u,j}}{\partial W^n} &= \frac{\partial J_{u,j}}{\partial l^N} \frac{\partial l^N}{\partial W^n} \\ &= \frac{\partial J_{u,j}}{\partial l_N} \frac{\partial f\left(W^N l^{N-1} + b^N\right)}{\partial \left(W^N l^{N-1} + b^N\right)} \frac{\partial \left(W^N l^{N-1} + b^N\right)}{\partial f\left(W^{N-1} l^{N-2} + b^{N-1}\right)} \cdots \\ &\quad \frac{\partial f\left(W^{n+1} l^n + b^{n+1}\right)}{\partial \left(W^{n+1} l^n + b^{n+1}\right)} \frac{\partial \left(W^{n+1} l^n + b^{n+1}\right)}{\partial f\left(W^n l^{n-1} + b^n\right)} \frac{\partial f\left(W^n l^{n-1} + b^n\right)}{\partial \left(W^n l^{n-1} + b^n\right)} l^{n-1} \end{aligned} \tag{7}$$

$$\begin{aligned} \frac{\partial J_{u,j}}{\partial b^n} &= \frac{\partial J_{u,j}}{\partial l^N} \frac{\partial l^N}{\partial b^n} \\ &= \frac{\partial J_{u,j}}{\partial l_N} \frac{\partial f\left(W^N l^{N-1} + b^N\right)}{\partial \left(W^N l^{N-1} + b^N\right)} \frac{\partial \left(W^N l^{N-1} + b^N\right)}{\partial f\left(W^{N-1} l^{N-2} + b^{N-1}\right)} \cdots \\ &\quad \frac{\partial f\left(W^{n+1} l^n + b^{n+1}\right)}{\partial \left(W^{n+1} l^n + b^{n+1}\right)} \frac{\partial \left(W^{n+1} l^n + b^{n+1}\right)}{\partial f\left(W^n l^{n-1} + b^n\right)} \frac{\partial f\left(W^n l^{n-1} + b^n\right)}{\partial \left(W^n l^{n-1} + b^n\right)} \end{aligned} \tag{8}$$

Let Δ^n be the term *w.r.t.* partial derivative at n-th layer as follows:

$$\Delta^n = \frac{\partial \left(W^{n+1} l^n + b^{n+1}\right)}{\partial f\left(W^n l^{n-1} + b^n\right)} \frac{\partial f\left(W^n l^{n-1} + b^n\right)}{\partial \left(W^n l^{n-1} + b^n\right)} \frac{l_{n-1}}{l_n}. \tag{9}$$

Then, $\partial J_{u,j}/\partial W^n$ and $\partial J_{u,j}/\partial b^n$ can be computed by error backpropagation from the last N-th layer to the n-th layer as follows [5], [6], [7]:

$$\begin{aligned} &\frac{\partial J_{u,j}}{\partial W^N} = \frac{\partial J_{u,j}}{\partial l^N} \frac{\partial l^N}{\partial W^N}, \quad \frac{\partial J_{u,j}}{\partial b^N} = \frac{\partial J_{u,j}}{\partial l^N} \frac{\partial l^N}{\partial b^N}, \\ &\frac{\partial l^N}{\partial W^N} = \frac{\partial f\left(W^N l^{N-1} + b^N\right)}{\partial \left(W^N l^{N-1} + b^N\right)} l^{N-1}, \quad \frac{\partial l^N}{\partial b^N} = \frac{\partial f\left(W^N l^{N-1} + b^N\right)}{\partial \left(W^N l^{N-1} + b^N\right)}, \\ &\frac{\partial l^N}{\partial W^n} = \frac{\partial l^N}{\partial W^{n+1}} \Delta^n, \quad \frac{\partial l^N}{\partial b^n} = \frac{\partial l^N}{\partial b^{n+1}} \Delta^n \frac{l_n}{l_{n-1}}, \end{aligned} \tag{10}$$

Therefore, we can finely control training process of each layer with different $\boldsymbol{\eta}$ and $\boldsymbol{\lambda}$ at layer-granularity based on (5), (6), (7), (8), (9), (10).

4.2.2 Part II: DE-Based Hyperparameter Auto-Learning

This part adopts DE to auto-learn Λ and H simultaneously, including Initialization, Mutation, Crossover, and Selection [18], [19], [20], [42], [43], [44], [45].

Initialization. At (t-1)-th training epoch, it starts from initializing a pair of Λ and H with *NP* individuals, where each pair is denoted as a 'target vector' as follows:

$$X_i = \{_i, \Lambda_i\} = \left\{ \left[\eta_i^1, \eta_i^2, .., \eta_i^N\right], \left[\lambda_i^1, \lambda_i^2, \ldots, \lambda_i^N\right] \right\}, \tag{11}$$

where X_i denotes the i-th individual, $i \in$ {1, 2, …, *NP*}. All the *NP* individuals form an original population.

Mutation. Each target vector X_i has a corresponding mutant vector V_i. The *DE/Rand/1* mutation strategy is adopted to generate V_i [18], [19], [20] as follows:

$$V_i = X_{r1} + F_i \cdot (X_{r2} - X_{r3}), \tag{12}$$

where $r1$, $r2$, $r3 \in$ {12,…,*NP*} are random and different from each other and i, and F_i is a scaling factor controlling the scaling of various vectors. Note that there are some other frequently adopted mutation strategies, including *DE/Best/1, DE/RandToBest/1, DE/Rand/2, DE/Best/2, and DE/RandToBest/2*. They are introduced in the Supplementary File, available online. In particular, F_i controls the balance of optimal performance and convergence speed. Previous scale factor local search in DE (SFLSDE) method [19], [20] is effective in making F_i self-adaptive as follows:

$$F_i = \begin{cases} SFGSS & if\, rand_3 < \tau_2 \\ SFHC & if\, \tau_2 \le rand_3 < \tau_3 \\ F_l + F_u \cdot rand_1 & if\, rand_2 < \tau_1 and\, rand_3 > \tau_3 \\ F_l + F_u \cdot rand_1 & if\, rand_2 \ge \tau_1 and\, rand_3 > \tau_3 \end{cases}, \tag{13}$$

where $rand_1$, $rand_2$, and $rand_3$ are uniformly random values between 0 and 1; $\boldsymbol{\tau}_1$, $\boldsymbol{\tau}_2$, and $\boldsymbol{\tau}_3$ are constant thresholds. According to [19], [20], all the parameters in (13) are recommended as follows: $SFGSS = 8$, $SFHC = 20$, $F_l = 0.1$, $F_u = 0.9$, $\boldsymbol{\tau}_1 = 0.1$, $\boldsymbol{\tau}_2 = 0.03$, $\boldsymbol{\tau}_3 = 0.07$, and F_i is initialized to a random value between 0 and 1.

Crossover. To increase population diversity, V_i is used to disturb X_i. Crossover operation is to generate a new trial vector U_i based on X_i and V_i. The *arithmetic crossover* strategy [18], [19], [20] is adopted to generate the trial vector U_i as follows:

$$U_i = X_i + \omega \cdot (V_i - X_i), \tag{14}$$

where $\boldsymbol{\omega}$ is a random number in [0, 1]. By combining (12) into (14), we have:

$$U_i = X_i + \omega \cdot (X_{r1} - X_i) + F_i \cdot (X_{r2} - X_{r3}). \tag{15}$$

Besides, the *binomial crossover* strategy is also usually used. It is introduced in the Supplementary File, available online.

Selection. Selection operation is to test whether U_i or X_i to make a DL-based recommendation model achieve better performance on the validation dataset as follows:

$$X_i = \begin{cases} U_i\, if\, perf(U_i) \ge perf(X_i) \\ X_i\, otherwise \end{cases}, \tag{16}$$

where *perf*(·) is a function that adopts a specific metric to evaluate a DL-based recommendation model's performance.

Next, DE-Opt moves to t-th training epoch after all the NP individuals (target vectors) are evolved, in which H and Λ can be set by any one of the evolved NP target vectors. Note that with such operations of part II, the auto-learned H and Λ can guarantee that the performance of the next training epoch is always better than, or at least the same to, the current epoch. Therefore, DE-Opt guarantees to boost the performance of a DL-based RS.

4.3 Theoretical Analysis and Proof

Theorem 1. *Given the searching space of $\boldsymbol{\eta}$ and $\boldsymbol{\lambda}$, DE-Opt is convergent in searching the values of $\boldsymbol{\eta}$ and $\boldsymbol{\lambda}$ to make a DL-based recommendation model perform better.*

***Proofs of theorem* 1.** To begin with, we introduce Markov chain of DE-Opt and convergence criteria. Then, we prove the convergence of DE-Opt.

4.3.1 Markov Chain of DE-Opt

Following previous studies [29]–[32], we build a discrete Markov chain for DE-Opt.

Initialization. From formula (11), the state of the population at time t can be described as $S(t) = \{X_{1,t}, X_{2,t}, \ldots, X_{NP,t}\} = \{(H_{1,t}, \Lambda_{1,t}), (H_{2,\,t}, \Lambda_{2,t}), \ldots, (H_{NP,t}, \Lambda_{NP,t})\}$.

Mutation. The purpose of the mutation operation is to generate a new individual based on the different individuals. According to formulas (12) and (13), we define the state transition probability of mutation operation as follows:

$$P_{X_{i,t}\rightarrow V_{i,t}} = \alpha_i, \left(0 \le \alpha_i \le \frac{1}{(NP-1)\cdot(NP-2)\cdot(NP-3)}\right), \tag{17}$$

Crossover. From formula (14), the probability of crossover operation is:

$$P_{(X_{i,t},V_{i,t})\rightarrow U_{i,t}} = k_i, (0 \le k_i \le 1). \tag{18}$$

Selection. From formula (16), the probability of selection operation generating evolved individual is:

$$P_{select} = \begin{cases} 1\, if\, perf(U_{i,t}) \ge perf(X_{i,t}) \\ 0\, otherwise \end{cases}. \tag{19}$$

Assuming that mutation, crossover, and selection operations of DE algorithm are independent of each other, and the selection process from three random individuals in mutation operation is also an independent random process. To sum up, the probability of individual transition from state $X_{i,t}$ to state $X_{i,t+1}$ in DE-Opt is:

$$P_{X_{i,t}\rightarrow X_{i,t+1}} = P_{X_{i,t}\rightarrow V_{i,t}} \cdot P_{(X_{i,t},V_{i,t})\rightarrow U_{i,t}} \cdot P_{select}. \tag{20}$$

From the above analyses, we note that for any two continuous population states set $S(t) = \{X_{1,t}, X_{2,t}, \ldots, X_{NP,t}\}$ and $S(t+1) = \{X_{1,t+1}, X_{2,t+1}, \ldots, X_{NP,t+1}\}$ in the population state sequence $\{S(t) \mid t \in N^+\}$, the state transition probability $P(S(t)\rightarrow S(t+1))$ is only related to the population state at time t. It is proved that the population state sequence has Markov characteristics. Since the population state set is discrete and finite, and the probability in formulas (17), (18) and (19) are independent of t, the population state sequence $\{S(t) \mid t \in N^+\}$ is a finite homogeneous Markov chain.

4.3.2 Convergence Criteria

For randomized algorithms, e.g., DE-Opt, the convergence criteria can be described as follows.

Definition 1. *Given a new iteration solution* $S(t+1) = \Phi(S(t), \boldsymbol{\xi})$ *of an optimization problem* $<I, perf(\cdot)>$, *where* $\Phi(\cdot)$ *is an optimization algorithm,* $\boldsymbol{\xi} = \{S(1), S(2), \ldots, S(t-1)\}$ *is previous searched solutions,* I *is the feasible solution space. For DE-Opt,* I *is the hyperparameters searching space and* $perf(\cdot)$ *is the fitness/performance function.*

Assumption 1. $perf(\Phi(S(t), \xi)) \ge perf(\xi)$, *s.t.* $perf(\Phi(S(t), \xi)) \ge perf(\xi)$, $\xi \in I$.

Definition 2. *Population state space is* $S = \{S(t) = \{X_{i,t} \mid X_{i,t} \in I, 1 \le i \le NP\}\}$, s.t. $S(t) = \{X_{i,t} \mid X_{i,t} \in I, 1 \le i \le NP\}$ *is the population state.*

Definition 3. *Global optimal state set is* $G = \{X \mid perf(X) = perf(g^*), X \in S\}$, s.t. g^* *is an optimal solution of* $<I, perf(\cdot)>$.

4.3.3 Convergence Proofs of DE-Opt

The DE-Opt can be represented as follows:

$$X_{i,t+1} = P_{X_{i,t}\rightarrow V_{i,t}} \cdot P_{(X_{i,t},V_{i,t})\rightarrow U_{i,t}} \cdot P_{select} \cdot X_{i,t}, \tag{21}$$

where $i = \{1, 2, \ldots, NP\}$. Formula (21) can be proved to be convergent as follows.

Lemma 1. *The DE-Opt satisfies* assumption *1, i.e.,* $perf(S(t+1)) \ge perf(S(t))$.

Proof. *According to selection operation, it is easy to proof that* $perf(X_{i,t+1}) \ge perf(X_{i,t})$. *From* Definitions *1 and 2, it is also easy to proof that* $perf(S(t+1)) \ge perf(S(t))$, *i.e., DE-Opt is monotonically increasing,* lemma *1 stands.*

Theorem 2. *The state sequence of DE-Opt* $\{S(t) \mid t \in N^+\}$ *is a finite homogeneous irreducible aperiodic Markov chain.*

Proof. *From the Markov chain of DE-Opt,* $\{S(t) \mid t \in N^+\}$ *has been proven to be a finite homogeneous Markov chain. From* Definition *2, we can obtain:*

$$\sum S(t+1) \in S \cdot P\{S(t) \cdot P_{X_{i,t}\rightarrow X_{i,t+1}} = S(t+1)\} = 1, \tag{22}$$

where $\forall S(t) \in S$, *therefore* $\{S(t) \mid t \in N^+\}$ *is irreducible Markov chain. For the following formula:*

$$\begin{aligned} &P(S(t) \cdot P_{X_{i,t}\rightarrow X_{i,t+1}} = S(t+1)) \\ &= \sum S \cdot P_{X_{i,t}\rightarrow V_{i,t}} \cdot P_{(X_{i,t},V_{i,t})\rightarrow U_{i,t}} \cdot P_{select}, \end{aligned} \tag{23}$$

there are

$$P_{X_{i,t}\rightarrow V_{i,t}} > 0, P_{(X_{i,t},V_{i,t})\rightarrow U_{i,t}} > 0, and P_{select} > 0, \tag{24}$$

therefore

$$P(X_{i,t} \cdot P_{X_{i,t}\rightarrow X_{i,t+1}} = X_{i,t+1}) > 0. \tag{25}$$

Consequently, $\{S(t) \mid t \in N^+\}$ is an aperiodic Markov chain.

Theorem 3. *In DE-Opt,* G *from* definition *3 is a closet set.*

TABLE 3
Algorithm DE-Opt

Steps	Operations
1	initializing
2	model construction
3	generate vectorized **H** and Λ as $\text{Population}_{\text{initialized}}$ for model
4	**while** epoch $t \leq \text{Epoch}_{\max}$ && not converge
5	calculate $\text{performance}_{\text{unevolved}}$ in the **validation** set
6	**for** i = 1 to NP
7	initialize X_i in Population_{t-1}
8	randomly select X_{r1}, X_{r2} and X_{r3} in Population_{t-1}
9	compute V_i by formula (12)
10	generate U_i from X_i and V_i by formula (15)
11	$H_{\text{train}} = U\ 1\ i$, $\Lambda_{\text{train}} = U\ 2\ i$
12	train model with H_{train} and Λ_{train} in the **training** set
13	calculate $\text{performance}_{\text{evolved}}$ in the **validation** set
14	Select X_i by formula (16)
15	update Population_{t-1} with X_i
16	end for
17	complete the evolution of Population_{t-1} to Population_t
18	t = t+1
19	end while

Proof. *From the Chapman-Kolmogorov equation [31], [32], transfer probability* Pl S(a),S(b) *from state* S(a) *to state* S(b) *after* l *transfer steps is:*

$$P^{l}_{S(a),S(b)} = \sum_{S(\varepsilon_1)\in S}\sum_{S(\varepsilon_2)\in S}\cdots\sum_{S(\varepsilon_{l-1})\in S} P(S(a)\cdot P_{X_{i,a}\to X_{i,r_1}} = S(\varepsilon_1))\cdot P(S(\varepsilon_1)\cdot P_{X_{i,\varepsilon_1}\to X_{i,\varepsilon_2}} = S(\varepsilon_2))\cdots P(S(\varepsilon_{l-1})\cdot P_{X_{i,\varepsilon_{l-1}}\to X_{i,b}} = S(b)),\ s.t.\forall l(l\geq 1), \forall S(a) = \{X_{1,a}, X_{2,a}, \cdots, X_{NP,a}\} \in G, \forall S(b) \notin G. \tag{26}$$

In each step, there is a transfer from $S(t-1) \in G$ to $S(t)$, and $t \in \{1, 2, \ldots, l\}$. There is $perf(S(t)) < perf(S(t\text{-}1)) = perf(g^*)$, *s.t.* $S(t-1) \in G$ and $S(t) \notin G$, therefore $P(S(t\text{-}1) \to S(t)) = 0$, i.e., $Pl\ S(a),S(b) = 0$. Consequently, G is a closed set on state space S.

Theorem 4. *No non-empty closed set* G′ *satisfies* $G' \cap G = \emptyset$ *on population state space* S *of the DE-Opt.*

Proof. *(Proof by contradiction). Given a non-empty closed set* G′ *on* S, $G' \cap G = \emptyset$, $S(a) = \{g^*_{1,a}, g^*_{2,a}, \ldots, g^*_{NP,a}\} \in G$, $\forall S(b) = \{X_{1,b}, X_{2,b}, \ldots, X_{NP,b}\} \in G'$, *there is* $\text{perf}(X_{i,b}) < \text{perf}(g^*\ i,a)$. *Thus, from formula (24),* $P(S(\varepsilon_{t+1}) \to S(\varepsilon_{t+a+1})) > 0$, $Pl\ S(a),S(b) > 0$, *which means that* G′ *is not a non-empty closed set.*

Theorem 5. *[31], [32]. Transfer probability* π_j *satisfies* $lim_{k\to\infty} p(S(k) = j) = \pi_j$ *when* $j \in E$, *and* $lim_{k\to\infty} p(S(k) = j) = 0$ *when* $j \notin E$, *s.t.* $ *non-empty closed sets* E *and* V *satisfy* $E \cap V = \emptyset$.

Theorem 6. *Let iterations tend to infinity,* S *will get into* G *in DE-Opt.*

Proof. *Theorems 2,3,4, and 5 ensure the authenticity of Theorem 6.*

Finally, from *Lemma* 1 and *Theorems* 2, 3, 4, 5, and 6, it is easy to prove that DE-Opt is convergent in the given searching space. Therefore, *Theorem* 1 holds.

4.4 Algorithm Design and Complexity Analysis

Based on the analyses of Section 4.2, we design *Algorithm DE-Opt*, with its pseudo-code presented in Table 3. We also analyze its time complexity in improving a DL-based RS as follow. Supposing the time complexity of an original model with fixed $\boldsymbol{\eta}$ and $\boldsymbol{\lambda}$ is $O(T_{model})$ and DE-Opt contains *NP* individuals. Then, the time complexity of DE-Opt is $O(T_{model}\times NP)$. Notably, an original model requires Grid Search to search optimal $\boldsymbol{\eta}$ and $\boldsymbol{\lambda}$. Assuming that each hyperparameter needs $\boldsymbol{\theta}$ times search, then the final time complexity of an original model is $O(T_{model}\times\boldsymbol{\theta}^2)$. In general, *NP* is set in a small range like from 5 to 20 (DE-Opt is 5, please refer to Section 5.5) [18], [19], [20], and each hyperparameter usually needs at least five times search (i.e., $\boldsymbol{\theta}\geq 5$) [8], [22]. Hence, the time complexity of DE-Opt is less than that of Grid Search in searching $\boldsymbol{\eta}$ and $\boldsymbol{\lambda}$ in general. Besides, DE-Opt can auto-learn the optimal $\boldsymbol{\eta}$ and $\boldsymbol{\lambda}$, which avoids the tedious Grid Search. In terms of space complexity, DE-Opt adopts *NP* arrays with length $2\times N$ to cache *NP* individuals, where each individual is a pair of Λ and H. Hence, the space complexity of DE-Opt is $O(N\times NP)$.

TABLE 4
Statistics of the Studied Datasets

No.	Name	#User	#Item	#Interaction	Density*
D1	Film Trust	1508	2071	37919	1.21%
D2	MovieLens 1M	6040	3706	1000209	4.47%
D3	Each Movie	72916	1628	2811983	2.36%

**Density denotes the percentage of observed entries in the dataset.*

5 Experiments

In the subsequent experiments, we aim at answering the following research questions (RQs):

- RQ.1. Can DE-Opt boost state-of-the-art recommendation models with fixed hyperparameters in both rating prediction and item ranking tasks?
- RQ.2. Does DE-Opt significantly outperform state-of-the-art hyperparameter searching competitors?
- RQ.3. How does DE-Opt empirically converge to optimal hyperparameter sets?
- RQ.4. How does the individual number *NP* of DE influence the performance of DE-Opt?
- RQ.5. How do mutation and crossover strategies of DE influence the performance of DE-Opt?

5.1 General Settings

Datasets. We adopt three real-world datasets [5] of *FilmTrust*[1], *MovieLens 1M*[1], and *Each Movie*[1] as the benchmark in the experiments. Their statistics are summarized in Table 4. We adopt the explicit ratings for the rating prediction task and transform them into binary code for the item ranking task.

Evaluation Metrics. Missing rating prediction and item ranking prediction (top-*K* recommendation) are two common tasks in an RS [5]. Considering missing rating prediction, mean absolute error (MAE) and root mean squared error (RMSE) are widely adopted as the evaluation metrics [5], [22] as follows:

$$MAE = \left(\sum_{(u,j)\in\Gamma} \left|y_{u,j} - \hat{y}_{u,j}\right|_{abs}\right) \Big/ |\Gamma|,$$
$$RMSE = \sqrt{\left(\sum_{(u,j)\in\Gamma} \left(y_{u,j} - \hat{y}_{u,j}\right)^2\right) \Big/ |\Gamma|},$$

where Γ denotes the testing set and $|\cdot|_{abs}$ calculates the absolute value of a given number. Note that a lower MAE/RMSE indicates a higher rating prediction accuracy.

TABLE 5
Summary of Comparison Models

Model	Description	
Fixed η and λ model	AutoRec [22]	A classic DL-based rating prediction model. It is a representative collaborative filtering method based on autoencoder. (WWW 2015)
	NRT [23]	It is a multitask learning with a rating prediction framework developed by combining multilayer perceptron and recurrent neural networks. (SIGIR 2017)
	MetaMF [24]	It is a federated learning framework for private rating prediction for mobile environments. (SIGIR 2020)
	NeuMF [14]	A classic DL-based item ranking model. It combines traditional matrix factorization and multilayer perceptron. (WWW 2017)
	LRML [25]	It is a neural architecture for collaborative ranking via memory-based attention. (WWW 2018)
	NGCF [21]	A GCN-based item ranking model. It exploits user-item graph structure by propagating. (SIGIR 2019)
η-adaptive model	Adam [17]	A classic SGD-based η-adaptive optimizer. It only requires first-order gradients with little memory for efficient stochastic optimization. (ICLR 2015)
	AMSGrad [12]	It is a variant of Adam that fixes the convergence issue and improves empirical performance. (ICLR 2018)
	AdaMod [13]	It proposes automatic warmup heuristic and long-term learning rate buffer to improve Adam. (arXiv 2019)
λ-adaptive model	SGDA [11]	It makes λ adaptive based on dimension-wise regularization and SGD. It is limited to adopt η-adaptive optimizers [8]. (WSDM 2012)
	λOpt [8]	It updates λ automatically and adaptively during training. It allows fine-grained regularization to build a better generalized model. (KDD 2019)
η-λ-adaptive model	λOpt- Adam	We adopt η-adaptive Adam to optimize λOpt.
	λOpt-AMSGrad	We adopt η-adaptive AMSGrad to optimize λOpt.
	λOpt-AdaMod	We adopt η-adaptive AdaMod to optimize λOpt.

TABLE 6
The Comparison Result of Rating Prediction Accuracy

Dataset	Metric	AutoRec		NRT		MetaMF	
		Grid Search	DE-Opt	Grid Search	DE-Opt	Grid Search	DE-Opt
D1	RMSE	**0.8223**	0.8225	0.8087	**0.8023**	0.8239	**0.8214**
	MAE	0.6188	**0.6179**	0.6259	**0.6124**	0.6469	**0.6379**
D2	RMSE	0.8532	**0.8520**	0.8791	**0.8771**	0.9370	**0.9090**
	MAE	0.6665	**0.6659**	0.6919	**0.6893**	0.7433	**0.7200**
D3	RMSE	0.3170	**0.3102**	**0.2594**	0.2599	0.2815	**0.2771**
	MAE	0.2370	**0.2359**	0.2021	**0.2018**	0.2240	**0.2197**
Statistics	*Win/Loss*	5/1	5/1	6/0			
	p-value*	**0.0313**	**0.0469**	**0.0156**			

A The accepted hypotheses with a significance level of 0.05 are highlighted.

On the other hand, item ranking prediction produces a ranked list with K items for recommendation, where K is a cutoff hyperparameter. Normalized discounted cumulative gain (NDCG), Recall, and hit ratio (HR) are widely adopted to evaluate the ranking prediction accuracy [8], [14]. Let $\Gamma_K(u)$ denote the intersection set between the top-K items generated by the tested model and the testing items of Γ on user u. Then, the NDCG of a tested model is computed as follows:

$$DCG@K(u,\hat{Y}) = \sum_{j=1}^{K} \frac{2^{\hat{y}_{u,j}} - 1}{\log_2(j+1)},$$

$$DCG@K(u,\Gamma) = \sum_{j=1}^{K} \frac{2^{y_{u,j}} - 1}{\log_2(j+1)},$$

$$NDCG@K = \frac{1}{|\Psi|} \sum_{u \in \Psi} \left(\frac{DCG@K(u,\hat{Y})}{DCG@K(u,\Gamma)} \right),$$

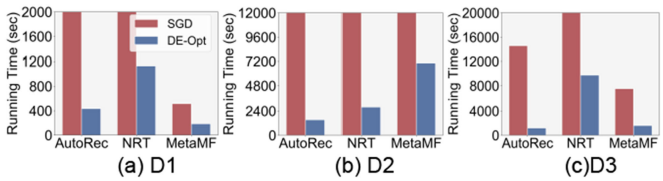


Fig. 4. The comparison results of GPU running time in rating prediction task.

TABLE 7
The Comparison Result of Item Ranking Accuracy

Dataset	Metric	NeuMF		LRML		NGCF	
		Grid Search	DE-Opt	Grid Search	DE-Opt	Grid Search	DE-Opt
D1	NDCG@10	0.4513	**0.4641**	**0.3781**	0.3746	0.3734	**0.3773**
	NDCG@20	0.5377	**0.5464**	**0.4588**	0.4556	0.4562	**0.4602**
	Recall@10	0.5404	**0.5500**	0.4385	**0.4460**	**0.4002**	0.3987
	Recall@20	0.7547	**0.7587**	0.6539	**0.6619**	**0.5778**	0.5775
D2	NDCG@10	0.3035	**0.3047**	0.2136	**0.2234**	0.3541	**0.3544**
	NDCG@20	0.2953	**0.2990**	0.2098	**0.2308**	0.3852	**0.3857**
	Recall@10	0.1241	**0.1262**	0.0806	**0.1296**	0.1366	**0.1392**
	Recall@20	0.1972	**0.2026**	0.1350	**0.2068**	0.2167	**0.2211**
D3	NDCG@10	0.1078	**0.1105**	0.0655	**0.0752**	**0.0523**	0.0528
	NDCG@20	0.1460	**0.1481**	0.0921	**0.1059**	0.0747	**0.0753**
	Recall@10	0.1908	**0.1921**	0.1138	**0.1315**	0.0651	**0.0663**
	Recall@20	**0.3158**	0.3147	0.2010	**0.2321**	0.1164	**0.1180**
Statistics	*Win/Loss*	11/1		10/2		9/3	
	p-value*	**0.0005**		**0.0012**		**0.0081**	

*A The accepted hypotheses with a significance level of 0.05 are highlighted.

where $\hat{y}_{u,j}$ denotes the j-th prediction in $\Gamma_K(u)$ in descending order, $y_{u,j}$ denotes the actual j-th item of Γ on user u in descending order, and Ψ denotes a set of contained users of Γ. The Recall of a tested model is computed as follows:

$$Recall@K = \frac{1}{|\Psi|}\sum_{u\in\Psi}\frac{|\Gamma_K(u)|}{K}.$$

If a test item appears in the recommended list, it is deemed as a hit. Then, the HR is calculated as follows:

$$hits@K(u) = \begin{cases} 1 & \Gamma_K(u) \notin \emptyset \\ 0 & \Gamma_K(u) \in \emptyset \end{cases}$$
$$HR@K = \frac{1}{|\Psi|}\sum_{u\in\Psi} hits@K(u).$$

Note that NDCG, Recall, and HR lie in the scale of [0, 1], where a higher value indicates a higher ranking prediction accuracy.

Baselines. We compare our DE-Opt with fourteen state-of-the-art models, including six fixed hyperparameter models and eight adaptive hyperparameter models. Their details are summarized in Table 5.

Implementation Detail. We set Grid Search and our DE-Opt to have the same searching space. Specially, we adopted the ten times amplification principle to set their searching space, i.e., the optimal values searched by Grid Search are amplified ten times for setting DE-Opt's searching space. Please refer to Table S.2–4 in the Supplementary File, available online, to see the details of the searching space. We randomly split the observed entries of each dataset into the training-validation-testing sets with 70%-10%-20% ratio, respectively. DE-Opt auto-learns the DL-based RSs' hyperparameters on the training set and the learning performances are supervised on the validation set, i.e., the the DL-based RSs' hyperparameters are auto-learned on the training set to make the resultant RS perform best on the validation set. Finally, all the trained models are tested on the testing set. All the other hyperparameters of tested models (except for $\boldsymbol{\lambda}$ and $\boldsymbol{\eta}$) are set by the original papers' recommendations. SGD is the default optimizer. Adaptive hyperparameter models have a basic deep matrix factorization module. The prediction rating and ranking list for a particular item are generated according to the original papers. We repeat each experiment five times and report average results.

System Configuration. The experiments are implemented by Python 3.7.0 with the libraries of Numpy, Tensorflow, Scipy, Sklearn, Pandas, and Torch. All experiments are run by 11G GTX1080ti GPUs. Our resource code is available at the following link: https://github.com/Wuziqiao/DE-Opt.

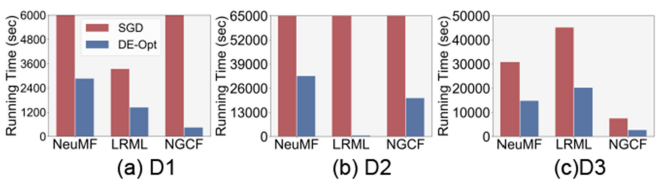


Fig. 5. The comparison results of GPU running time in item ranking task.

5.2 Comparison With Fixed Hyperparameter Model (RQ. 1)

5.2.1 Rating Prediction Task

Table 6 presents the comparison results of rating prediction, where we observe that DE-Opt can achieve lower RMSE/MAE than Grid Search method except for only one case. To check whether DE-Opt significantly outperforms Grid Search method, we conduct the Wilcoxon signed-ranks test [26] on the comparisons of RMSE/MAE of Table 6. The p-value denotes the significance level, and if it is lower than a significance level 0.05, the hypotheses are accepted. The results demonstrate that DE-Opt can significantly improve AutoRec, NRT, and MetaMF in rating prediction accuracy.

Besides, Fig. 4 records the GPU running time of each model tested on all the datasets. We observe that Grid Search significantly consumes more running time than DE-Opt. The reason is that DE-Opt can auto-learn the appropriate hyperparameters with only one-time training, which avoids the multiple times training with Grid Search.

TABLE 8
The Comparison Results with Adaptive Hyperparameter Models

Dataset	Metrics	η-adaptive(λ-GridSearch)			λ-adaptive (η-GridSearch)		η-λ-adaptive			DE-Opt
		Adam	AMSGrad	AdaMod	SGDA	λOpt	λOpt-Adam	λOpt-AMSGrad	λOpt-AdaMod	
D1	NDCG@10	0.4321•	0.4412•	0.4311•	0.4115•	**0.4466**	0.4332•	0.4401•	0.4342•	0.4438
	NDCG@20	0.4924•	0.4809•	0.4723•	0.4660•	0.3955•	0.4651•	0.4863•	0.4628•	**0.5061**
	HR@10	**0.6446**	0.6370•	0.6324•	0.5907•	0.5995•	0.6329•	0.6407•	0.6350•	0.6419
	HR@20	0.7920•	0.7891•	0.7787•	0.7404•	0.6800•	0.7778•	0.7840•	0.7814•	**0.8044**
D2	NDCG@10	0.0482•	0.0474•	0.0476•	0.0031•	0.0238•	0.0852•	**0.1095**	0.0863•	0.1050
	NDCG@20	0.0627•	0.0560•	0.0642•	0.0074•	0.0323•	0.1033•	0.1318•	0.1048•	**0.1340**
	HR@10	0.0965•	0.0951•	0.0970•	0.0069•	0.0482•	0.1594•	**0.1959**	0.1611•	0.1907
	HR@20	0.1579•	0.1421•	0.1595•	0.0225•	0.0831•	0.2370•	0.2826•	0.2402•	**0.2847**
D3	NDCG@10	0.0697•	0.0557•	0.0689•	0.0297•	0.0222•	0.0614•	0.0659•	0.0621•	**0.0778**
	NDCG@20	0.0967•	0.0973•	0.0987•	0.0417•	0.0362•	0.0859•	0.0943•	0.0913•	**0.1013**
	HR@10	0.1478•	0.1183•	0.1474•	0.0624•	0.0495•	0.1309•	0.1407•	0.1315•	**0.1640**
	HR@20	0.2593•	0.2579•	0.2609•	0.1097•	0.1059•	0.2327•	0.2546•	0.2453•	**0.2701**
Statistics	Win/Loss	11/1	12/0	12/0	12/0	11/1	12/0	10/2	12/0	92/4○
	p-value*	**0.0005**	**0.0002**	**0.0002**	**0.0002**	**0.0005**	**0.0002**	**0.0134**	**0.0002**	–
	F-rank◊	6.17	4.75	5.25	1.67	2.08	4.33	6.83	5.25	**8.67**

• *Indicates DE-Opt outperforms the comparison models.* ○ *The total Win/Loss cases of DE-Opt.* * *The accepted hypotheses with a significance level of 0.05 are highlighted.* ◊ *A higher F-rank value denotes a higher item ranking accuracy.*

5.2.2 Item Ranking Task

The accuracy comparison results of item ranking are shown in Table 7. They are also analyzed with the Wilcoxon signed-ranks test [26] with a significance level 0.05. Similar to rating prediction results, the results validate that DE-Opt can significantly boost NeuMF, LRML, and NGCF in item ranking. Besides, the comparison results of GPU running time of item ranking are also shown in Fig. 5. The results are also similar to rating prediction results, i.e., DE-Opt consumes significantly less GPU running time than Grid Search.

5.3 Comparison With Adaptive Hyperparameter Model (RQ. 2)

The accuracy comparison results of item ranking are shown in Table 8. We also adopt the Wilcoxon signed-ranks test [26] to check these comparison results with a significance level 0.05. Besides, we further make the Friedman test [26]. It is to simultaneously compare the performance of multiple models on multiple datasets by the F-rank value. From Table 8, we have the important findings:

- DE-Opt achieves the higher ranking prediction accuracy on most cases. It loses the comparison with only 3 cases. Specifically, the total win/ loss cases situation is 92/4.
- All the p-values are smaller than 0.05, which indicates that DE-Opt has significantly higher ranking prediction accuracy than all the adaptive hyperparameter models with a significance level 0.05.
- DE-Opt has the highest F-rank value among all the models, which further verifies that it has the overall highest ranking prediction accuracy on the tested datasets.

Besides, the computational efficiency is also compared. Fig. 6 shows the GPU running time of all the involved models, where we note that DE-Opt consumes significantly less running time than the other adaptive hyperparameter models.

Therefore, the above experimental results demonstrate that DE-Opt significantly outperforms the adaptive hyperparameter models in terms of both prediction accuracy and computational efficiency.

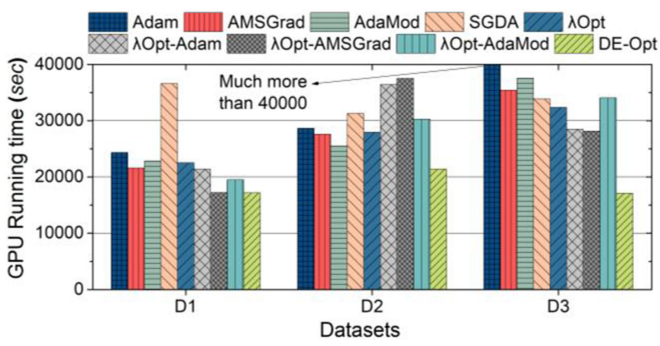


Fig. 6. The comparison results of GPU running time between DE-Opt and adaptive hyperparameter models.

5.4 Auto-Learning the Hyperparameters (RQ. 3)

To check how DE-Opt auto-learns λ and η, we draw their change curves during its training process. A deep matrix factorization is also adopted as the basic model of DE-Opt. Figs. 7 and 8 show the results on all the datasets, where we find that both λ and η oscillate at the beginning of training. Since DE-Opt tries to search for better hyperparameters in the initial steps, the oscillation can prompt hyperparameters to jump out of the poor local optimal solutions. As the training epoch increases, they gradually stabilize in a certain range. The above results well prove that DE-Opt can make a DL-based recommendation model' λ and η be simultaneously auto-learned at layer-granularity during its training process. Besides, we also check the descent/ascent curves of loss/accuracy of the tested model during the training

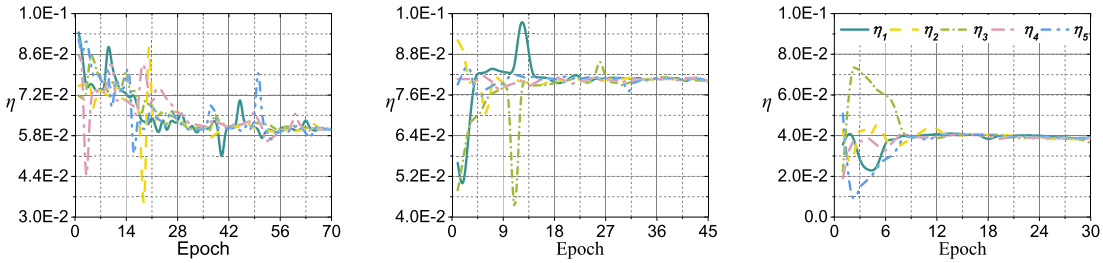


Fig. 7. Auto-learning trajectories of η with five individuals on all the datasets.

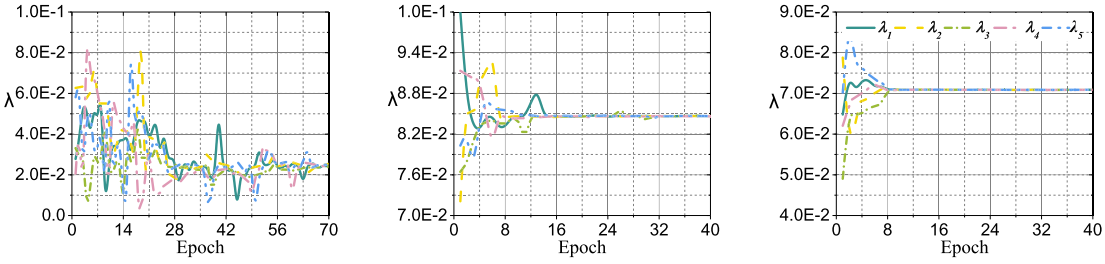


Fig. 8. Auto-learning trajectories of λ with five individuals on all the dataset.

process. Please refer to Figs. S.1–3 in the Supplementary File, available online, to see the details. The results also prove that DE-Opt can make the loss/accuracy of the tested model better with more training epochs until convergence.

5.5 Influence of Individual Number NP (RQ. 4)

To evaluate the influences of individual number *NP* to DE-Opt, we carry out experiments with different *NP* settings. The results are presented in Figs. 9 and 10. We note that DE-Opt performs relatively robustly and well when *NP* is between 5 to 15. Subsequently, as *NP* continues to increase, the NDCG and HR decreases in general. The reason may be that more individuals (larger *NP*) cause more random processes during the training process. Overall, DE-Opt is relatively robust to the individual number *NP*. We set $NP = 5$ for our DE-Opt.

5.6 Influences of Mutation and Crossover Strategies (RQ. 5)

This sub-section analyzes the influences of different mutation and crossover strategies to DE-Opt. Concretely, we

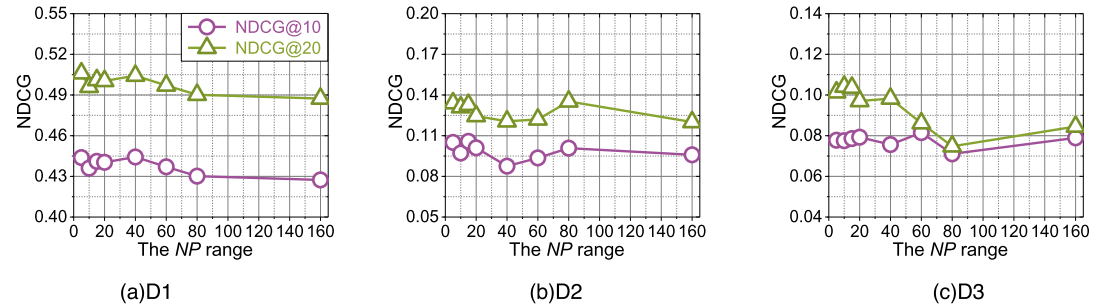


Fig. 9. The influence of *NP* on NDCG on all the dataset.

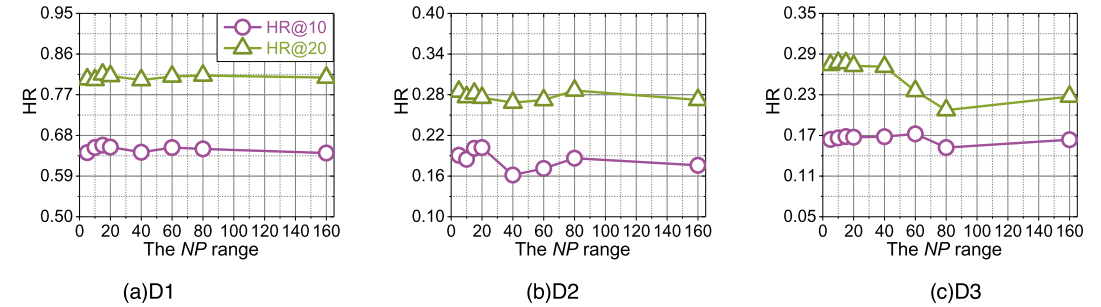


Fig. 10. The influence of *NP* on HR on all the dataset.

TABLE 9
The Cmparison Rsult With SX Mtation Srategies in NDCG and HR

Dataset	Metric	DE/Rand/1	DE/Best/1	DE/RandToBest/1	DE/Best/2	DE/Rand/2	DE/RandToBest/2
D1	NDCG@10	0.4438•	0.4504•	**0.4777**	0.4538•	0.4439•	0.4613
	NDCG@20	0.5061•	0.5057•	0.5021•	0.5044•	0.5076•	**0.5170**
	HR@10	0.6419•	0.6450•	0.6503•	0.6511•	0.6461•	**0.6615**
	HR@20	0.8044•	0.8084•	0.8127•	0.8140•	0.7988•	**0.8228**
D2	NDCG@10	0.1050•	0.1034•	**0.1098**	0.1097	0.0956•	0.1053
	NDCG@20	0.1340•	0.1375•	0.1387•	0.1353•	0.1285•	**0.1390**
	HR@10	0.1907•	0.1917•	0.2010	**0.2028**	0.1916•	0.1998
	HR@20	0.2847•	**0.2945**	0.2838•	0.2857•	0.2825•	0.2858
D3	NDCG@10	0.0778•	0.0785•	0.0753•	0.0759	0.0728•	**0.0786**
	NDCG@20	0.1013•	0.1032•	**0.1088**	0.1078•	0.1060•	0.1078
	HR@10	0.1640•	0.1640•	0.1612•	0.1634•	0.1530•	**0.1705**
	HR@20	0.2701•	0.2724•	0.2794•	0.2730•	0.2658•	**0.2819**
Statistics	*Win/Loss*	12/0	11/1	8/4	9/3	12/0	52/8○
	p-value*	**0.0002**	**0.0046**	0.1331	**0.0147**	**0.0002**	–
	F-rank◊	2.375	3.375	4.0	4.042	1.83	**5.375**

compare the six classical mutation strategies [42], [43], [44], [45], i.e., DE/Rand/1 (adopted in Section 4.2), DE/Best/1, DE/RandToBest/1, DE/Best/2, DE/Rand/2, and DE/RandToBest/2; and the three situations of crossover strategies [42], [43], [44], [45], i.e., without crossover, binomial crossover, and arithmetic crossover (adopted in Section 4.2). Next, we mainly analyze how these strategies influence the prediction accuracy of DE-Opt. Note that the influence of computational efficiency is not significant, we move the results to Table S.5 in the Supplementary File, available online. Besides, the details of these mutation and crossover strategies are also moved to Section S.IV in the Supplementary File, available online.

5.6.1 Influences of Mutation Strategies

Table 9 records the comparison results of different mutation strategies on all the datasets, where we observe that DE/RandToBest/2 seems to perform best. To further validate this observation, we conduct the statistics of win/loss counts, the Wilcoxon signed-ranks test, and the Friedman test by comparing DE/RandToBest/2 with the other five mutation strategies. The statistical results are also recorded in Table 9, where we note that: 1) DE/RandToBest/2 evidently wins the other mutation strategies on most cases, 2) DE/RandToBest/2 significantly outperforms the other mutation strategies except for DE/RandToBest/1, and 3) DE/RandToBest/2 performs best among all the mutation strategies as it achieves the highest F-rank value.

TABLE 10
The Cmparison Rsult of Iem Rnking Acuracy

Dataset	Metric	WithOut Crossover	Binomial Crossover	Arithmetic Crossover
D1	NDCG@10	0.3898•	0.4067•	**0.4438**
	NDCG@20	0.4386•	0.4661•	**0.5061**
	Recall@10	0.5603•	0.6190•	**0.6419**
	Recall@20	0.7325•	0.7647•	**0.8044**
D2	NDCG@10	0.0917•	**0.1080**	0.1050
	NDCG@20	0.1312•	0.1189•	**0.1340**
	Recall@10	0.1706•	0.1896•	**0.1907**
	Recall@20	0.2829•	0.2596•	**0.2847**
D3	NDCG@10	0.0461•	0.0712•	**0.0778**
	NDCG@20	0.0924•	**0.1041**	0.1013
	Recall@10	0.1019•	0.1538•	**0.1640**
	Recall@20	0.2570•	0.2618•	**0.2701**
Statistics	*Win/Loss*	12/0	10/2	22/2○
	p-value*	**0.0002**	**0.0024**	–
	F-rank◊	1.177	2	**2.833**

5.6.2 Influences of Crossover Strategies

Table 10 presents the comparison results of different crossover strategies on all the datasets. We also conduct the statistics of win/loss counts, the Wilcoxon signed-ranks test, and the Friedman test by comparing arithmetic crossover with the other two strategies. The statistical results evidently show that arithmetic crossover significantly outperforms the other two strategies as the *p*-values are smaller than 0.05, and it achieves the highest F-rank value.

5.6.3 Summary

By analyzing the different mutation and crossover strategies, we empirically demonstrate that DE/RandToBest/2 mutation and arithmetic crossover strategies make DE-Opt perform better than the other strategies. Therefore, we recommend adopting DE/RandToBest/2 mutation and arithmetic crossover strategies for DE-Opt in practical applications.

6 Conclusion

This paper proposes a general hyperparameter optimization framework based on differential evolution (DE) for existing deep learning (DL)-based recommender systems (RSs), named DE-Opt. Its main idea is to incorporate DE into a DL-based RS's training process to auto-learn its hyperparameters $\boldsymbol{\lambda}$ (regularization coefficient) and $\boldsymbol{\eta}$ (learning rate) at once. DE-Opt allows to yield optimal hyperparameter pairs at a layer-granularity, where each layer has different $\boldsymbol{\lambda}$ and $\boldsymbol{\eta}$, thereby uplifting the resultant recommendation performance. The empirical evaluation of DE-Opt was benchmarked on three widely-used datasets with extensive experiments. The results substantiate that: 1) DE-Opt is compatible with six state-of-the-art DL-based recommendation models and significantly improves their

performance while making their $\boldsymbol{\lambda}$ and $\boldsymbol{\eta}$ adaptive, and 2) DE-Opt significantly outperforms its competitors whose $\boldsymbol{\lambda}$ and/or $\boldsymbol{\eta}$ are adaptative. In the future, we will extend DE-Opt to more advanced RS models to evaluate its performance in complex scenarios, such as agricultural RS [47].

REFERENCES

[1] Y. Gu, K. Vyas, M. Shen, J. Yang, and G. Yang, "Deep graph-based multimodal feature embedding for endomicroscopy image retrieval," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 2, pp. 481–492, Feb. 2021.

[2] L. Wu, X. He, X. Wang, K. Zhang, and M. Wang, "A survey on accuracy-oriented neural recommendation: From collaborative filtering to information-rich recommendation," *IEEE Trans. Knowl. Data Eng.*, to be published, doi: 10.1109/TKDE.2022.3145690.

[3] D. Wang, X. Zhang, D. Yu, G. Xu, and S. Deng, "CAME: Content- and context-aware music embedding for recommendation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 3, pp. 1375–1388, Mar. 2021.

[4] L. Ren and W. Wang, "A granular SVM-based method for top-N web services recommendation," *IEEE Trans. Serv. Comput.*, vol. 15, no. 1, pp. 457–469, Jan./Feb. 2022.

[5] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Comput. Surv.*, vol. 52, no. 1, pp. 5:1–5:38, 2019.

[6] L. Huang, Y. Ma, S. Wang, and Y. Liu, "An attention-based spatiotemporal LSTM network for next POI recommendation," *IEEE Trans. Serv. Comput.*, vol. 14, no. 6, pp. 1585–1597, Nov./Dec. 2021.

[7] M. Maher and S. Sakr, "Smartml: A meta learning-based framework for automated selection and hyperparameter tuning for machine learning algorithms," in *Proc. 22nd Int. Conf. Extending Database Technol.*, 2019, pp. 554–557.

[8] Y. Chen et al., "λopt: Learn to regularize recommender models in finer levels," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 978–986.

[9] X. Dong, J. Shen, W. Wang, L. Shao, H. Ling, and F. Porikli, "Dynamical hyperparameter optimization via deep reinforcement learning in tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 5, pp. 1515–1529, May 2021.

[10] E. Ndiaye, T. Le, O. Fercoq, J. Salmon, and I. Takeuchi, "Safe grid search with optimal complexity," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 4771–4780.

[11] S. Rendle, "Learning recommender systems with adaptive regularization," in *Proc. 15th Int. Conf. Web Search Web Data Mining*, 2012, pp. 133–142.

[12] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," in *Proc. 6th Int. Conf. Learn. Representations*, 2018, pp. 1–23.

[13] J. Ding, X. Ren, R. Luo, and X. Sun, "An adaptive and momental bound method for stochastic learning," *CoRR*, vol. abs/1910.12249, 2019. [Online]. Available: http://arxiv.org/abs/1910.12249

[14] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua, "Neural collaborative filtering," in *Proc. 26th Int. Conf. World Wide Web*, 2017, pp. 173–182.

[15] Z. Fu et al., "Deep learning for search and recommender systems in practice," in *Proc. 26th ACM SIGKDD Conf. Knowl. Discov. Data Mining Virtual Event*, 2020, pp. 3515–3516.

[16] H. Lee, J. Im, S. Jang, H. Cho, and S. Chung, "Melu: Meta-learned user preference estimator for cold-start recommendation," in *Proc. 25th ACM SIGKDD Conf. Knowl. Discov. Data Mining*, 2019, pp. 1073–1082.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015, pp. 1–15.

[18] Bilal, M. P., H. Zaheer, L. García-Hernández, and A. Abraham, "Differential evolution: A review of more than two decades of research," *Eng. Appl. Artif. Intell.*, vol. 90, 2020, Art. no. 103479.

[19] I. Triguero, S. García, and F. Herrera, "SEG-SSC: A framework based on synthetic examples generation for self-labeled semi-supervised classification," *IEEE Trans. Cybern.*, vol. 45, no. 4, pp. 622–634, Apr. 2015.

[20] F. Neri and V. Tirronen, "Scale factor local search in differential evolution," *Memetic Comput.*, vol. 1, no. 2, pp. 153–171, 2009.

[21] X. Wang, X. He, M. Wang, F. Feng, and T. Chua, "Neural graph collaborative filtering," in *Proc. 42nd Int. Conf. Res. Develop. Inf. Retrieval*, 2019, pp. 165–174.

[22] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie, "Autorec: Autoencoders meet collaborative filtering," in *Proc. 24th Int. Conf. World Wide Web Companion*, 2015, pp. 111–112.

[23] P. Li, Z. Wang, Z. Ren, L. Bing, and W. Lam, "Neural rating regression with abstractive tips generation for recommendation," in *Proc. 40th Int. Conf. Res. Develop. Inf. Retrieval*, 2017, pp. 345–354.

[24] Y. Lin et al., "Meta matrix factorization for federated rating predictions," in *Proc. 43rd Int. Conf. Res. Develop. Inf. Retrieval*, 2020, pp. 981–990.

[25] Y. Tay, L. A. Tuan, and S. C. Hui, "Latent relational metric learning via memory-based attention for collaborative ranking," in *Proc. World Wide Web Conf. World Wide Web*, 2018, pp. 729–739.

[26] J. Demsar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006.

[27] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, "Lightgcn: Simplifying and powering graph convolution network for recommendation," in *Proc. 43rd Int. Conf. Res. Develop. Inf. Retrieval*, 2020, pp. 639–648.

[28] X. Wang, R. Wang, C. Shi, G. Song, and Q. Li, "Multi-component graph convolutional collaborative filtering," in *Proc. 34th AAAI Conf. Artif. Intell.*, 2020. pp. 6267–6274.

[29] G. Xu and G. Yu, "On convergence analysis of particle swarm optimization algorithm," *J. Comput. Appl. Math.*, vol. 333, pp. 65–73, 2018.

[30] J. Kwon, "Robust visual tracking based on variational auto-encoding Markov chain monte carlo," *Inf. Sci.*, vol. 512, pp. 1308–1323, 2020.

[31] Q. Li and M. Shang, "BALFA: A brain storm optimization-based adaptive latent factor analysis model," *Inf. Sci.*, vol. 578, pp. 913–929, 2021.

[32] R. Poli and W. B. Langdon, "Markov chain models of bare-bones particle swarm optimizers," in *Proc. Genet. Evol. Computation Conf.*, 2007, pp. 142–149.

[33] S. Roy, S. Sridharan, S. Jain, and A. Raghunathan, "TxSim: Modeling training of deep neural networks on resistive crossbar systems," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 29, no. 4, pp. 730–738, Apr. 2021.

[34] M. Wang, W. Fu, X. He, S. Hao, and X. Wu, "A survey on large-scale machine learning," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 6, pp. 2574–2594, Jun. 2022, doi: 10.1109/TKDE.2020.3015777.

[35] R. Yu et al., "Personalized adaptive meta learning for cold-start user preference prediction," in *Proc. 35th AAAI Conf. Artif. Intell.*, 2021. pp. 10 772–10 780.

[36] D. Wu, Q. He, X. Luo, M. Shang, Y. He, and G. Wang, "A posterior-neighborhood-regularized latent factor model for highly accurate web service QoS prediction," *IEEE Trans. Serv. Comput.*, vol. 15, no. 2, pp. 793–805, Mar./Apr. 2022.

[37] D. Wu, M. Shang, X. Luo, and Z. Wang, "An Ll-and-L2-norm-oriented latent factor model for recommender systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 10, pp. 5775–5788, Oct. 2022.

[38] R. P. Parouha and P. Verma, "State-of-the-art reviews of meta-heuristic algorithms with their novel proposal for unconstrained optimization and applications," *Arch. Comput. Methods Eng.*, vol. 28, pp. 4049–4115, 2021.

[39] D. Gong, Z. Zhang, Q. Shi, A. van den Hengel, C. Shen, and Y. Zhang, "Learning deep gradient descent optimization for image deconvolution," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 12, pp. 5468–5482, Dec. 2020.

[40] Q. Zhang, L. Cao, C. Shi, and Z. Niu, "Neural time-aware sequential recommendation by jointly modeling preference dynamics and explicit feature couplings," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 10, pp. 5125–5137, Oct. 2022, doi: 10.1109/TNNLS.2021.3069058.

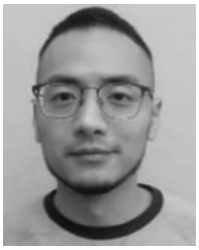
[41] W.-D. Xi, L. Huang, C.-D. Wang, Y.-Y. Zheng, and J.-H. Lai, "Deep rating and review neural network for item recommendation," *IEEE Trans. Neural Netw. Learn. Syst*, vol. 33, no. 11, pp. 6726–6736, Nov. 2022, doi: 10.1109/TNNLS.2021.3083264.

[42] H. R. Chamorro, I. Riano, R. Gerndt, I. Zelinka, F. Gonzalez-Longatt, and V. K. Sood, "Synthetic inertia control based on fuzzy adaptive differential evolution," *Int. J. Elect. Power Energy Syst.*, vol. 105, pp. 803–813, 2019.

[43] S. Prabha and R. Yadav, "Differential evolution with biological-based mutation operator," *Eng. Sci. Technol. Int. J.*, vol. 23, no. 2, pp. 253–263, 2020.

[44] S. Li, Q. Gu, W. Gong, and B. Ning, "An enhanced adaptive differential evolution algorithm for parameter extraction of photovoltaic models," *Energy Convers. Manage.*, vol. 205, 2020, Art. no. 112443.

[45] N. B. Guedria, "An accelerated differential evolution algorithm with new operators for multi-damage detection in plate-like structures," *Appl. Math. Modelling*, vol. 80, pp. 366–383, 2020.
[46] M. Garanayak, S. N. Mohanty, A. K. Jagadev, and S. Sahoo, "Recommender system using item based collaborative filtering (CF) and K-means," *Int. J. Knowl.-Based Intell. Eng. Syst.*, vol. 23, no. 2, pp. 93–101, 2019.
[47] M. Garanayak, G. Sahu, S. N. Mohanty, and A. K. Jagadev, "Agricultural recommendation system for crops using different machine learning regression methods," *Int. J. Agricultural Environ. Inf. Syst.*, vol. 12, no. 1, pp. 1–20, 2021.



Di Wu (Member, IEEE) received the PhD degree from the Chongqing Institute of Green and IntelligentnTechnology (CIGIT), Chinese Academy of Sciences (CAS), China, in 2019, and then joined CIGIT, CAS, China. He is currently a professor with the College of Computer and Information Science, Southwest University, Chongqing, China. He has more than 50 publications, including 12 IEEE Transactions papers of *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Transactions on Neural Networks and Learning Systems*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Systems, Man, and Cybernetics*, etc., and conferences of AAAI, WWW, ICDM, IJCAI, etc. He is servicing as an *associate editor for Frontiers in Neurorobotics* (SCI, IF2.65). Homepage: https://wudi1989.github.io/Homepage/.

Bo Sun received the BS degree in optoelectronic information science and engineering from Tianjin University, Tianjin, China, in 2019. He is currently working toward the MS degree in computer technology with the University of Chinese Academy of Sciences (UCAS), Beijing, China. His research interests include data mining and deep learning.

Mingsheng Shang received the PhD degree in computer science from the University of Electronic Science and Technology of China (UESTC). He joined the Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, China, in 2015, and is currently a professor of computer science and engineering. He has more than 100 publications including *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Transactions on Neural Networks and Learning Systems*, *IEEE Transactions on Cybernetics*, *IEEE Transactions on Systems, Man, and Cybernetics*, *IEEE Transactions on Services Computing*, etc. His research interests include data mining, complex networks, cloud computing, and their applications.